A Complexity Tale: Web Configurators

Gilles Perrouin
PReCISE Research Center
University of Namur, Belgium
gilles.perrouin@unamur.be

Mathieu Acher
IRISA, University of Rennes I, France
mathieu.acher@irisa.fr

Jean-Marc Davril
PReCISE Research Center,
University of Namur, Belgium
jean-marc.davril@unamur.be

Axel Legay
INRIA Rennes, France
axel.legay@inria.fr

Patrick Heymans
PReCISE Research Center,
University of Namur, Belgium
patrick.heymans@unamur.be

ABSTRACT

Online configurators are basically everywhere. From physical goods (cars, clothes) to services (cloud solutions, insurances, etc.) such configurators have pervaded many areas of everyday life, in order to provide the customers products tailored to their needs. Being sometimes the only interfaces between product suppliers and consumers, much care has been devoted to the HCI aspects of configurators, aiming at offering an enjoyable buying experience. However, at the backend, the management of numerous and complex configuration options results from ad-hoc process rather than a systematic variability-aware engineering approach. We present our experience in analysing web configurators and formalising configuration options in terms of feature models or product configuration matrices. We also consider behavioural issues and perspectives on their architectural design.

CCS Concepts

•**Software and its engineering** → **Software product lines; Software reverse engineering;** *Software configuration management and version control systems;*

Keywords

Web Configurator, Complexity, Variability

1. WEB CONFIGURATORS

Many B2B and B2C companies provide their customers with Web configurators. The purpose of these configurators is to assist stakeholders in configuring a product or service that will meet their requirements and expectations.

Our experience with industrial partners, as well as our empirical study of 111 Web configurators [1], reveal that configurators are commonly developed in an ad hoc manner, which can raise reliability and maintenance issues. From a security perspective, a naive implementation can be problematic as well [4] (e.g., users can bypass a configurator to get access to features or copyrighted data).

The root cause of these issues is the complexity encountered in the variability of the configurable products. A complex variability translates into a large number of configuration options and constraints among them. As a result organizations face difficulties for eliciting, formalizing, realizing, and maintaining configurators (e.g., variability crosscuts different artefacts, constraints can be hard-coded in the control-flow).

In this paper we perform a short retrospective of what has been investigated so far. We consider three major axes – reverse engineering, behavioural analysis, and forward engineering of configurators – and we discuss some of the underlying challenges. We present model-driven engineering approaches to address them and manage complexity.

The remainder of the paper is structured as follows. Section 2 presents reverse engineering techniques to help practitioners recover variability models from existing artefacts. This is a key step towards the re-engineering of legacy systems into more reliable model-driven configurators. Section 3 presents techniques and formalisms for behavioural analysis. We discuss the need of model-driven techniques capable validating configuration tasks or configurator evolution, possibly at run-time. Section 4 proposes a generic architecture for deriving web configurators from models. Section 5 summarizes the different problems and challenges to undertake for taming the complexity of configurators.

2. FORMALISING CONFIGURATION OPTIONS

2.1 Re-Engineering Legacy Configurators

While web configurators are commonly developed in an ad hoc manner [1], configurators engineered from variability models are easier to maintain, to test and are more reliable. This is especially true when the number of configuration options becomes large and when the configurator is required to enforce many complex configuration constraints. In order to address the widespread absence of model-driven solutions observed in web configurators, we proposed an approach based on feature modeling for re-engineering legacy online configurators [10].

Feature Models (FM) were introduced 25 years ago as part of the FODA (Feature Oriented Domain Analysis) method [27]. FMs are tree-like diagrams which purpose is to specify the variability among the products of a product line. A FM hierarchically decomposes the features of its products by using boolean operators. FMs have been extended and formalised through formal semantics [13, 17, 30] and are now equipped with comprehensive tool support [8, 28]. The formal semantics of FMs can be used to support the execution of configuration tasks [21] and to generate configurators' Graphical User Interfaces (GUI) [11].

The re-engineering approach in [10] proposes to recover *variability patterns* from existing web configurators and syn-

thesize them into an FM that can be used to engineer a new configurator. In this approach, we analyze the GUI of an existing configurator and classify graphical widgets (e.g. HTML or jQuery elements and images) into configuration elements (e.g. option, description field, and constraint). As an example, radio buttons are commonly used to ensure that an end-user can only select one single configuration option from a group of several options. Radio buttons can thus be mapped to a variability pattern expressing a mutual exclusion between the options. The set of all the extracted variability patterns can be encoded into an FM that will then be used to engineer the new configurator.

2.2 Reverse-Engineering Variability Models from Heterogenous Data

In addition to support the (re)engineering of configurators, FMs can also be useful for conducting configurator's domain analysis tasks. Domain knowledge about the products, their commonalities/variabilities, and their features is often scattered through heterogeneous types of documents. While some of these documents can provide structured or semi-structured data (e.g. list of products specifications) others are unstructured and require prior processing (e.g. textual description of products). Large collections of product documentation artefacts call for the development of automatic techniques to help practitioners recover the scattered variability information.

Existing works address the synthesis of FMs from both product specifications [5, 6, 18, 24, 31, 32] and collections of textual documents [19, 20, 23, 34].

These techniques typically use an intermediate (formal) representation before actually performing the synthesis of the model. For instance, a formula representing the set of products/configurations can be used. The gathering of a formula can be a difficult task and involves static or dynamic source code analysis [31], mining features' co-occurrences [19], etc. Then satisfiability techniques are applied on the formula to compute variability information. As many different FMs can be built from the same formula, the extraction requires heuristics for guiding the selection of the proper organization of features (i.e., feature hierarchy, feature groups, attribute placement) [5, 6].

Another possible approach is to use a product comparison matrix (PCM) as an intermediate representation. A PCM is a tabular product-by-feature representation that describes the list of possible product configurations in terms of features. PCMs can be directly found on the web (e.g., in configurators or comparators). PCMs can also be extracted from a collection of documents written in natural language [7]. Once the product configurations are formalized into a PCM, a synthesis algorithm can be applied to recover variability patterns among the features and construct an FM (see, e.g., [6, 18]).

3. BEHAVIOURAL ANALYSIS

3.1 Feature Configuration Workflows

Configuring a complex product requires expertise. Indeed, dependencies between certain options may requires that a given sequence of choices is followed in order to avoid redoing some of them. The situation is even more complicated when several users (because they own expertise on only one part of the configuration options) need to interact with the configurator. To organise such a configuration process, feature configuration workflows were proposed [2, 25, 26]. The idea is to explicitly model the configuration process in a workflow, where tasks corresponds to set of choices relevant for a given stakeholder of the configurator. Relying on the formal underpinning of workflows [33], it is possible to guide the user through configuration steps and perform soundness analyses and ensure that each option have been covered at the end of the process.

3.2 Analysing Configured Products

Variability also induces complexity in the behaviour of configured products. Let us take the example of the well-known Tesla electric cars. Tesla offers only two different models (S and X). Hardware variability is limited to different battery capacities, transmission options, car colors. But these cars have the particularity to be configurable by the user even once the car is delivered: the autopilot/self-parking feature is downloaded and activated by the owner. In terms of configuration, this implies that owners continuously configures their cars, not only prior to ordering them. This "feature on demand" approach [12], obviously raises challenges in terms of security and safety of the configured products, as these features have the ability to change configured products behaviour.

To address this situation, we need to couple configuration models with verification ones, and we believe that formalisms such as *featured transition systems* (FTSs) [14], aka transition systems tagged with features, are appropriate. FTSs enable to concisely model the product(s) that execute a given behaviour and variability-aware model checkers have been developed for them [16]. Of course, as for software product line verification where this formalism comes from, variability-aware configurators are relevant for TESLA when it incorporates a new feature to its catalogue. Additionally since these cars dynamically adapt - the behaviour of the car is itself variability-aware and context-dependent - verifying if the introduction of a new feature is safe, may require that the configurator of the car itself embeds its FTS and model-checker. To be efficient, on-the-fly reduction techniques of the verification space must be employed: for example, Kim *et al.* prune statically configurations that cannot violate a given property, reducing the number of configurations to monitor at runtime [29]. Cordy *et al.* have proposed incremental verification of software product lines to deal with partial configurations [15], though this technique has not been extended to run-time scenarios yet. Thus, both run-time and design time verification techniques are needed in this case.

This scenario illustrates the complexity web configurators have or will have to face, calling for dedicated architectural solutions. In the next section, we present an architectural pattern for such configurators to harness complexity and discuss perspective on their engineering.

4. MODEL-BASED ENGINEERING

4.1 An architectural Pattern for Web Configurators

Model-driven engineering techniques can provide the right formalisms and abstractions to manage the multi-faceted complexity of configurators. Figure 1 presents a generic model-based architecture for web configurators.

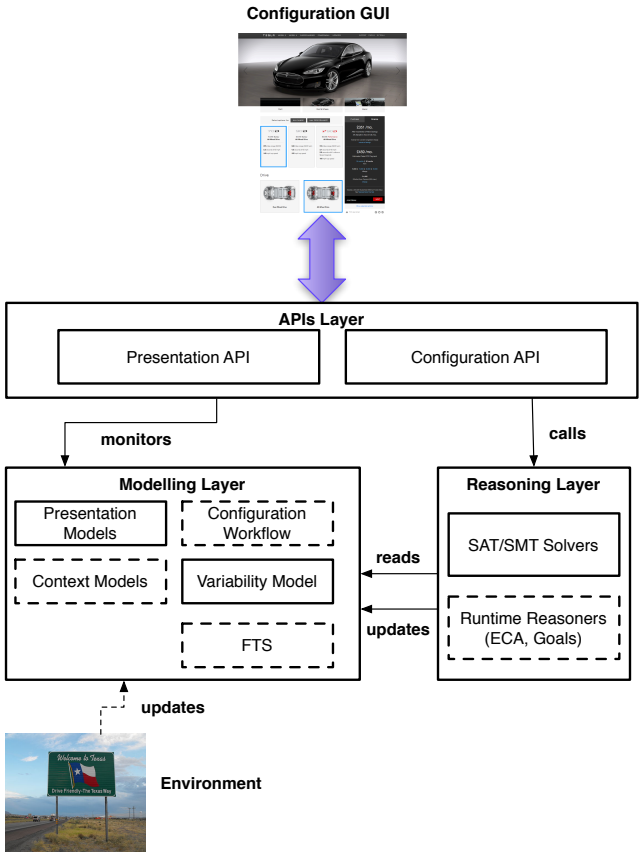

Figure 1: A Generic Web Configurator Architecture

The architecture is decomposed in three layers: *(i)* the APIs layer offers high-level services such as knowing the consequences of a choice realized by the user or updates the interface to hide options that are no longer available, *(ii)* the modelling layer that contains all the models required for the operation of the configurator and *(iii)* the reasoning layer that contains solving and adaptation engines. In the modelling layer, a FM formally describes the options and constraints of the configurator. The presentation models contain views, beautification (such as CSS sheets, styles, etc. [9]) that drive the generation of the configurator GUI. Models and tools represented in frames with dashed lines are optional. We already mentioned behavioural models roles. Context models reflect the environment. They support scenarios in which a given set of options should be deactivated because of the environment: e.g. sudden failure of the battery implies energy-saving mode by turning off multimedia or air conditioning. Runtime adaptation is often realized through Event Condition Action (ECA) rules or goal-based optimisation of a quantitative property. Such properties are also relevant for recommendation: it is well-known that some applications are weather-dependent: e.g; food, clothes, etc. A configurator aware of this can therefore rank configurations accordingly. This run-time information may be combined with recommendations obtained by mining product descriptions [22], and we believe that PCMs form a suited formalism to support recommendation computations.

4.2 Perspective: Configurator Integration

While the design of complex configurators can be facilitated by the use of model-based techniques and efficient design/run-time analysis, one aspect should not be neglected: configurators seldom work in isolation. Indeed, configurators must integrate with product databases to provide price information, warehouses/shops to compute the number of remaining products, 3D engines to provide visual rendering of products, or 3D printers to build configured products directly [3]. Given the heterogeneity of systems configurators have to connect to, modular configurator architectures with explicit interfaces are key to manage this diversity.

5. CONCLUSION

We presented problems and existing solutions for addressing the complexity of web configurators. The large number of options and constraints practitioners have to handle typically induces maintenance, reliability, security and integration issues.

Three research axes have proposed: reverse engineering, behavioural analysis, and forward engineering. These topics are still subject to active investigation; here are a few examples of possible research directions. From a reverse engineering perspective, the use of dynamic methods to explore and mine the configuration space is promising but their computational cost can be prohibitive. More heuristics and empirical studies are needed to further understand the limits and possible trade-offs. From a behavioural analysis perspective, the integration of different verification mechanisms is a difficult challenge, demanding in term of model expressiveness and hybrid analyses (quantitative/qualitative). Finally, forward engineering challenges include the systematic use of generative techniques for various configurators' parts and dealing with integration issues, particularly for e-commerce web configurators.

Acknowledgements

We would like to thank the organisers to have invited us to present our views in this paper. This research was partly funded by Walloon region under the INOGRAMS project (n°7171). Jean-Marc Davril is supported by the FNRS under a FRIA doctoral grant.

6. REFERENCES

[1] E. K. Abbasi, A. Hubaux, M. Acher, Q. Boucher, and P. Heymans. The anatomy of a sales configurator: An empirical study of 111 cases. In *CAISE*, volume 7908 of *Lecture Notes in Computer Science*, pages 162–177, Valencia, Spain, June 2013. Springer.

[2] E. K. Abbasi, A. Hubaux, and P. Heymans. A toolset for feature-based configuration workflows. In *SPLC*, pages 65–69. IEEE, 2011.

[3] M. Acher, B. Baudry, O. Barais, and J.-M. Jézéquel. Customization and 3d printing: A challenging playground for software product lines. In *SPLC (Vol. 1)*, pages 142–146. ACM, 2014.

[4] M. Acher, G. Bécan, B. Combemale, B. Baudry, and J.-M. Jézéquel. Product lines can jeopardize their trade secrets. In *ESEC/FSE*, pages 930–933, New York, NY, USA, 2015. ACM.

[5] G. Bécan, M. Acher, B. Baudry, and S. Ben Nasr. Breathing ontological knowledge into feature model

synthesis: An empirical study. *Empirical Software Engineering (ESE)*, 2015.

[6] G. Bécan, R. Behjati, A. Gotlieb, and M. Acher. Synthesis of attributed feature models from product descriptions. In *SPLC (Vol. 1)*, pages 1–10, Nashville, TN, USA, jul 2015. ACM.

[7] S. Ben Nasr, G. Bécan, M. Acher, J. a. B. Ferreira Filho, B. Baudry, N. Sannier, and J.-M. Davril. Matrixminer: A red pill to architect informal product descriptions in the matrix. In *ESEC/FSE*, ESEC/FSE 2015, pages 982–985. ACM, 2015.

[8] D. Beuche. Modeling and building software product lines with pure:: variants. In *SPLC*, pages 255–255. ACM, 2012.

[9] Q. Boucher. *Engineering Configuration Graphical User Interfaces from Variability Models*. PhD thesis, University of Namur, September 2014.

[10] Q. Boucher, E. K. Abbasi, A. Hubaux, G. Perrouin, M. Acher, and P. Heymans. Towards more reliable configurators: a re-engineering perspective. In *PLEASE*, pages 29–32. IEEE/ACM, 2012.

[11] Q. Boucher, G. Perrouin, and P. Heymans. Deriving configuration interfaces from feature models: A vision paper. In *VaMoS*, pages 37–44. ACM, 2012.

[12] N. Cardozo, W. D. Meuter, K. Mens, S. González, and P. Orban. Features on demand. In *VaMoS*, pages 18:1–18:8. ACM, 2014.

[13] A. Classen, Q. Boucher, and P. Heymans. A text-based approach to feature modelling: Syntax and semantics of tvl. *Science of Computer Programming*, 76(12):1130–1143, 2011.

[14] A. Classen, M. Cordy, P.-Y. Schobbens, P. Heymans, A. Legay, and J.-F. Raskin. Featured transition systems: Foundations for verifying variability-intensive systems and their application to ltl model checking. *IEEE TSE*, 39(8):1069–1089, 2013.

[15] M. Cordy, P.-Y. Schobbens, P. Heymans, and A. Legay. Towards an incremental automata-based approach for software product-line model checking. In *SPLC (Vol. 2)*, SPLC '12, pages 74–81. ACM, 2012.

[16] M. Cordy, M. Willemart, B. Dawagne, P. Heymans, and P. Schobbens. An extensible platform for product-line behavioural analysis. In *SPLC Workshops (Vol. 2)*, pages 102–109. ACM, 2014.

[17] K. Czarnecki, S. Helsen, and U. Eisenecker. Formalizing cardinality-based feature models and their specialization. *Software process: Improvement and practice*, 10(1):7–29, 2005.

[18] K. Czarnecki, S. She, and A. Wasowski. Sample spaces and feature models: There and back again. In *SPLC*. IEEE, 2008.

[19] J. Davril, E. Delfosse, N. Hariri, M. Acher, J. Cleland-Huang, and P. Heymans. Feature model extraction from large collections of informal product descriptions. In *ESEC/FSE*, pages 290–300. ACM, 2013.

[20] J.-M. Davril, M. Cordy, P. Heymans, and M. Acher. Using fuzzy modeling for consistent definitions of product qualities in requirements. In *IEEE 2nd Workshop on Artificial Intelligence for Requirements Engineering (AIRE'15)*, Ottawa , Canada, aug 2015.

[21] D. Dhungana, A. Falkner, and A. Haselböck. Configuration of cardinality-based feature models using generative constraint satisfaction. In *37th EUROMICRO SEAA*, pages 100–103. IEEE, 2011.

[22] H. Dumitru, M. Gibiec, N. Hariri, J. Cleland-Huang, B. Mobasher, C. Castro-Herrera, and M. Mirakhorli. On-demand feature recommendations derived from mining public product descriptions. In *ICSE*, pages 181–190. IEEE/ACM, May 2011.

[23] A. Ferrari, G. O. Spagnolo, and F. Dell'Orletta. Mining commonalities and variabilities from natural language documents. In *SPLC*. ACM, 2013.

[24] E. N. Haslinger, R. E. Lopez-Herrejon, and A. Egyed. On extracting feature models from sets of valid feature combinations. In *FASE*, pages 53–67. Springer, 2013.

[25] A. Hubaux, A. Classen, and P. Heymans. Formal modelling of feature configuration workflows. In *SPLC*, volume 446, pages 221–230. ACM, 2009.

[26] A. Hubaux, P. Heymans, P. Schobbens, D. Deridder, and E. K. Abbasi. Supporting multiple perspectives in feature-based configuration. *Software and System Modeling*, 12(3):641–663, 2013.

[27] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-oriented domain analysis (foda) feasibility study. Technical report, DTIC Document, 1990.

[28] C. Kastner, T. Thum, G. Saake, J. Feigenspan, T. Leich, F. Wielgorz, and S. Apel. Featureide: A tool framework for feature-oriented software development. In *ICSE*, pages 611–614. IEEE Computer Society, 2009.

[29] C. H. P. Kim, E. Bodden, D. S. Batory, and S. Khurshid. Reducing configurations to monitor in a software product line. In *Runtime Verification*, volume 6418 of *Lecture Notes in Computer Science*, pages 285–299. Springer, 2010.

[30] P.-Y. Schobbens, P. Heymans, J.-C. Trigaux, and Y. Bontemps. Generic semantics of feature diagrams. *Computer Networks*, 51(2):456–479, 2007.

[31] S. She, R. Lotufo, T. Berger, A. Wasowski, and K. Czarnecki. Reverse engineering feature models. In *ICSE*, pages 461–470. IEEE, 2011.

[32] S. She, U. Ryssel, N. Andersen, A. Wasowski, and K. Czarnecki. Efficient synthesis of feature models. *Information and Software Technology*, 56(9):1122–1143, 2014.

[33] W. van der Aalst and A. ter Hofstede. Yawl: yet another workflow language. *Information Systems*, 30(4):245 – 275, 2005.

[34] N. Weston, R. Chitchyan, and A. Rashid. A framework for constructing semantically composable feature models from natural language requirements. In *SPLC*, pages 211–220. ACM, 2009.